文章编号:1001 - 9081(2007)01 - 0219 - 02

多实例工作流模式的 π 演算形式化

梁爱南,李长云,黄贤明
(湖南工业大学 计算机科学与技术系,湖南 株洲 412008)
(shuruo688@21cn.com)

摘 要:多实例工作流模式是一类重要的工作流模式。π 演算是一种移动进程代数,可用于对并发和动态变化的系统进行建模。对 π 演算进行了研究,提出了以 π 演算作为工作流形式化的基础,利用 π 演算对多实例工作流模式进行了详细的描述。

关键词:π 演算;ECA 规则;多实例工作流模式
中图分类号: TP311.52 **文献标识码**:A

Formalizing multiple instance workflow patterns based on the π-calculus

LIANG Ai-nan, LI Chang-yun, HUANG Xian-ming
(*Department of Computer, Hunan university of Technology, Zhuzhou Hunan* 412008, *China*)

Abstract: Multiple instance workflow patterns are important workflow patterns. The π-calculus is a kind of mobile process algebra which can be used to model concurrent and dynamic systems. Having done the research of the π-calculus, the π-calculus was proposed as a formal foundation for workflow, furthermore multiple instance workflow patterns were described by using the π-calculus in detail.

Key words: π-calculus; ECA rule; multiple instance workflow patterns

0 引言

为了实现工作流管理功能,我们必须将业务过程从现实世界中抽象出来,并用一种形式化方法对其进行描述。π 演算(π-calculus)是 Robin Milner 提出的以进程间的移动通信为研究重点的并发理论,它是对 CCS(Calculus of Communication System)的发展[1,2]。π 演算是系统交互行为建模的理论基础,适合描述动态系统。它既提供了行为等价理论,又支持系统行为的分析。在工作流建模阶段使用 π 演算有助于清楚地描述工作流;而在模型建立后,则可利用 π 演算来推演系统的行为,同时验证模型的正确性,如发现系统行为不完整、死锁、缺少同步等;另外,π 演算作为一种强大和成熟的形式化方法,π 演算有支持其正确性验证和相关应用的工具,所以 π 演算自然地成为工作流过程建模的理想工具,例如,文献[3,4]使用 π 演算对工作流进行了初步的形式化。

工作流模式是工作流过程建模的基本构造单元,工作流模式从控制流的角度系统地描述了过程定义语言需要满足的业务需求,与特定的工作流语言无关。多实例工作流模式是一类重要的工作流模式,它为工作流过程处理带来了极大的灵活性。使用 π 演算来形式化多实例工作流模式,能提高模型语义的精确性,并且具有较强的表达能力,实现较简洁。本文着眼于对多实例工作流模式进行 π 演算形式化。

1 π 演算

在 π 演算中,进程是并发运行实体的单位,并以名字来统一定义通道名以及在通道中传送的消息,每个进程都有若干与其他进程联系的通道,数据结构在这里被封装为与特定通道相关的进程,外部进程通过这些通道来操作相关结构。其基本计算实体为名字和进程,进程之间的通信是通过传递名字来完成。由于 π 演算不但可以传递 CCS 中的变量和值等,还可以传递通道名,并且将这几种实体都统称为名字而不再作区分,这使得 π 演算具有了建立新通道的能力,因此 π 演算可以用来描述结构不断变化的并发系统。

π 演算有几种不同的符号表示[1,2],下面介绍 π 演算的基本语法,它可由以下 BNF 范式给出:

$P ::= M \mid P \mid P \mid vzP \mid !P$

$M ::= 0 \mid \pi.P \mid M + M$

$\pi ::= \bar{x}<y> \mid x(z) \mid \tau \mid [x = y]\pi$

π 演算中最简单的实体是名字,进程通过名字进行交互,并在交互中传递名字。名字的语法定义是标识符。上述 π 演算语法定义中,x 是单个名字,$\bar{x}$ 与 x 互为对偶名字,进程通过对偶名字进行交互。P 是 π 演算中另一实体进程的语法定义,而 M 是 π 演算中"和"(summations)的语法定义,用以表达选择关系。π 的语法给出进程能够执行的四种动作,称为前缀。进程通过执行这些动作而进行演化。

π 演算语法的形式语义运用约简关系和变迁关系定义,下面运用自然语言给出它的直观定义。

(1) 0 表示非活动进程,即不做任何工作的进程。

(2) 前缀 $\pi.P$ 表示具有 π 表示的单一行为能力,该能力执行后,执行进程 P。

输出前缀 $\bar{x}<y>.p$ 表示通过名字 x 输出名字 y,然后执行进程 P。

输入前缀 $x(z).p$ 表示通过名字 x 输入名字,并用输入的名字替换进程 P 中 z,然后执行替换后的进程 P。

收稿日期:2006 - 07 - 24;**修订日期**:2006 - 12 - 07 **基金项目**:湖南省自然科学基金资助项目(05JJ30122);湖南省教育厅科研项目(052520);湖南省教育厅优秀青年项目(06B023);湖南工业大学博士基金

作者简介:梁爱南(1976 -),女,湖南湘潭人,讲师,硕士研究生,主要研究方向:工作流; 李长云(1972 -),湖南耒阳人,男,副教授,博士,主要研究方向:工作流、语义 Web、软件体系结构; 黄贤明(1976 -),男,湖南常德人,讲师,主要研究方向:数据库.

哑前缀 $\tau.P$ 表示做哑动作 τ 然后执行 P，一般来说，τ 用于表示进程外部不可见的内部动作。

匹配前缀 $[x=y]\pi.P$ 表示当 x 和 y 是同一名字时，执行 $\pi.P$，否则不做任何工作。

（3）“和”$M1+M2$ 表示选择执行进程 $M1$、$M2$ 中的一个。

（4）组装 $P1 \mid P2$ 表示并行执行 $P1$ 和 $P2$。

（5）限制 vzP 表示名字 z 是 P 的局部名字，P 在名字 z 上的外部动作被禁止，但 P 通过名字 z 的内部通信是允许的。

（6）复制 $!P$ 表示无限多个 P 的组装。

2 用 π 演算形式化多实例工作流模式

2.1 多实例工作流模式简介

工作流技术将应用逻辑与过程逻辑分离开来，过程逻辑是通过工作流语言描述的。为了提供基本的工作流建模并评价工作流语言的描述能力，文献[5]借鉴设计模式的思想，对工作流模式进行了收集和研究，文中提到了四种多实例工作流模式。

多实例工作流模式是指在一个过程实例的执行轨迹中，过程模型中的同一个活动执行了多次，从而产生了多个活动实例的情况。多实例工作流模式最大的特点在于模型中的活动与实际执行的活动实例存在着一对多的关系。文献[5]中提到的四种多实例工作流模式如下：

（1）没有同步的多实例：在一个工作流实例的环境中，一个活动的多个实例可以被创建。也就是说在控制线程外，产生新的线程。这些线程相互独立，不需要同步。例如：在客户订书的流程中，客户可以对不同的书下单，因此订书的活动存在多个实例，订 A 书的实例，订 B 书的实例等。

（2）具有先验设计时知识的多实例：在一个流程实例中，一个活动被实例化的次数在设计时是可知的。一旦所有已知的实例完成，其他的活动需要被执行。例如：某个订单，需要三次不同的审核。

（3）具有先验运行时知识的多实例：在一个流程实例中，一个活动被实例的话，次数在设计时是不可知的，它依赖于流程特性或者资源可用性，仅在流程实例运行时的某个阶段可知。例如：在一个评审团对科学论文的评审流程中，评审的动作依赖于论文的内容，评审团人员的数量，评审团的信誉。

（4）不具有先验运行时知识的多实例：在一个流程实例中，一个活动被实例的话，次数在设计时是不可知的，在流程运行的过程中也是不可知的。例如：在运输 100 台计算机的流程中，每次运输的计算机的个数是不可知的，因此整个的运输次数也是不可知的。

其中，(2)、(3)、(4)属于同步的多实例模式。这三种有一个共同的特点：在一个过程实例中可以多次激发一个活动，当该活动的所有实例都完成之后才能开始后继的其他活动。

2.2 多实例工作流模式的 π 演算形式化方法

用 π 演算对工作流模式进行形式化时将活动模型化为进程，让每一个活动对应一个独立的进程，每一个进程有前置条件和后置条件。进程 P 是进程 Q 的前置条件表示 P 完成后，Q 才能开始；进程 P 是进程 Q 的后置条件表示 Q 完成后，Q 要给 P 发送消息。对进程定义时使用事件—条件—动作（Event-Condition-Action，ECA）规则，ECA 规则描述了触发活动的事件和内部条件，实际上也描述了执行活动的执行依赖关系。ECA 规则中的事件和条件对应进程的前置条件，事件模型化为 π 演算中的输入前缀，条件模型化为 π 演算中的匹配前缀；ECA 规则中的动作分成两种：分别对应进程的内部动作和进程的后置条件。显然，如果一个进程定义时没有事件部分（即输入前缀），则这个进程表示一个开始活动；如果一个进程定义时没有后置条件，则这个进程表示一个最后的活动。因此，定义一个基本活动可用如下 π 演算形式：

$x.[a=b].\tau.\bar{y}.0$

其中，输入前缀 x 对应 ECA 规则中的事件，匹配前缀 $[a=b]$ 对应ECA规则中的条件，$\tau.\bar{y}.0$ 对应 ECA 规则中的动作。

下面使用上述方法形式化上文提到的 4 种多实例工作流模式：

1）没有同步的多实例模式的解决办法

对这种模式进行 π 演算形式化时，我们把一个活动 A 模型化为进程 A，另一个活动 B 模型化为进程 B，进程 A 和进程 B 通过通道 b 进行通信，进程 B 的多个实例可由进程 A 的多个复制产生，如图 1。由上文介绍的 π 演算形式化工作流模式的基本思想，可得到如下的 π 演算形式化表示：

$A=\tau_A.!\bar{b}.0$

$B=!b.\tau_B.B'$

其中，进程 B 在完成内部动作 τ_B 后，激发后续进程 B'。

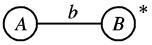

图 1 没有同步的多实例模式

2）具有先验设计时知识的多实例模式的解决办法

对这种模式进行 π 演算形式化时，我们把一个活动 A 模型化为进程 A，另外两个活动 B 和 C 模型化为进程 B 和进程 C，进程 A 和进程 B 通过通道 b 进行通信，进程 B 和进程 C 通过通道 c 进行通信。进程 B 的多个实例（实例个数在设计时可知）可由进程 A 的多个复制产生，只有当进程 B 的多个实例执行完成后才能开始进程 C 中的动作，如图 2。由上文介绍的 π 演算形式化工作流模式的基本思想，可得到如下的 π 演算形式化表示（先以 B 有 3 个实例同步为例）：

$A=\tau_A.\bar{b}.\bar{b}.\bar{b}.0$

$B=!b.\tau_B.\bar{c}.0$

$C=c.c.c.\tau_C.C'$

其中，进程 C 在完成内部动作 τ_C 后，激发后续进程 C'。

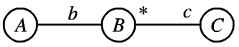

图 2 具有先验设计时知识的多实例模式

对于 B 有 n 个实例同步的情况，可用如下形式表示：

$A \mid B \mid C \equiv \tau_A.\{\bar{b}\}_1^n.0 \mid !b.\tau_B.\bar{c}.0 \mid \{c\}_1^n.\tau_C.C'$

3）具有先验运行时知识的多实例模式的解决办法

类似于具有先验设计时知识的多实例模式的解决办法，也可用图2表示，但是，B 的实例个数在设计时是不可知的，在 A 过程运行时活动 B 被实例化之前的某个阶段可获知，这时要引入一个进程 A 和进程 B 私有的名字 $start$，可得到如下的 π 演算形式化表示：

$A=(\upsilon run)\tau_A.A_1(c) \mid run.!\overline{start}.0$

$A_1(x)=(\upsilon y)\bar{b}<y>.y<x>.A_1(y)+\overline{run}.\bar{x}.0$

$B=!b(y).y(x).start.\tau_B.y.\bar{x}.0$

$C=c.\tau_C.C'$

这种模式运行时类似于一个如下的动态链：

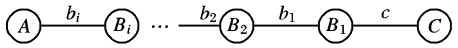

图 3 具有先验运行时知识的多实例模式的动态链

进程A首先拥有一个后继进程C的名字c，然后进程B的

（下转第 224 页）

n_{junk} 取 1 500：

$$SP = \frac{n_{junk\to junk}}{n_{junk\to junk} + n_{legit\to junk}}$$

$$SR = \frac{n_{junk\to junk}}{n_{junk}}$$

表 2 采用 10 次交叉验证方法得到的结果

属性个数	SP	SR	$S_{junk\to junk}$	$S_{legit\to junk}$
100	0.933	0.788	1 182	85
200	0.966	0.814	1 221	43
500	0.965	0.818	1 227	45

表 3 加入规则后得到的结果

属性个数	SP	SR	$S_{junk\to junk}$	$S_{legit\to junk}$
100	0.97	0.889	1 333	37
200	0.985	0.939	1 409	22
500	0.985	0.933	1 400	21

表 4 加入长度判断后得到的结果

属性个数	SP	SR	$S_{junk\to junk}$	$S_{legit\to junk}$
100	0.988	0.954	1 431	18
200	0.991	0.963	1 445	13
500	0.991	0.964	1 446	13

5.2 垃圾短信发送者识别

我们共从代理者收到垃圾短信号码汇报的个数为 35 个，其中确认垃圾发送者 34 个，而且这 1 个非垃圾发送者 f 值最小，这说明公式 f 对识别垃圾发送者有帮助。

5.3 性能测试

表 5 系统的性能分析

属性个数	分词时间/s	分类时间/s	内存消耗/MB
100	2.3	0.01	2.7
300	2.4	0.01	2.8
500	2.3	0.01	2.9

贝叶斯分类计算速度很快，影响处理速度的是分词算法。我们在 dopod 535 移动电话上进行了测试。dopod 535 系统运行 Windows Mobile phone。它的 CPU 为 200MHz 的 ARM 微处理器，32MB内存。当系统发现某短信为垃圾短信时，会在短信内容前标记“垃圾”两个字。表 5 给出了系统的性能分析。它显示属性的个数对性能影响很小，一般手机的运算速度和内存完全能满足贝叶斯过滤算法的需要。

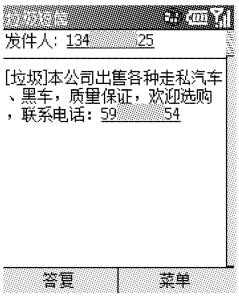

图 3 测试界面

6 结语

基于贝叶斯的分布式短信过滤系统具有自学习和更新能力，因此它能克服传统过滤器容易过时的问题。文章还提供了一种对垃圾短信发送者进行排名的方法。实验表明，我们提供的系统能够以较高的准确率识别垃圾信息，并且更适合于移动环境。

随着彩信的普及，基于彩信的垃圾信息也出现了，如何有效的过滤彩信将是我们下一步的工作。

参考文献：

[1] PAUL G. A plan for spam[EB/OL]. Http://www.Paulgraham.com/spam.html, 2002-12-10.

[2] PATRICK P, DEKAN GL. Spam Cop: A spam classification & organization program[A]. Proceedings of AAAI Workshop on Learning for Text Categorization[C]. 1998.

[3] MEHRAN S, SUSAN D, DAVID H. A Bayesian approach to filtering junk Email[A]. Proceedings of AAAI Workshop on Learning for Text Categorization[C]. 1998.

[4] PAUL G. Better Bayesian filtering[EB/OL]. Http://www.paulgraham.com/better.html, 2003-12-10.

[5] DOMINGOS P, PAZZANI M. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier[A]. Proceeding of the 13th International Conference on Machine Learning[C]. 1996.

[6] LANGLEY P, WAYNE I, THOMPSON K. An Analysis of Bayesian Classifiers[A]. Proceeding of the 10th National Conference[C]. 1992.

(上接第 220 页)

多个实例使用递归不断插入在 A 和 C 中。

4）不具有先验运行时知识的多实例模式的解决办法

这与具有先验运行时知识的多实例模式的解决办法很相似，不同的是这种模式中实例化的次数预先不可知，在需要时就创建新实例，直到不需要时为止。由此，不要使用名字 *start* 和 *run*，可得到如下的 π 演算形式化表示：

$$A = \tau_A.A_1(c)$$

$$A_1(x) = (\upsilon y)\bar{b} < y >.y < x >.A_1(y) + \bar{x}.0$$

$$B = !b(y).y(x).\tau_B.y.\bar{x}.0$$

$$C = c.\tau_C.C'$$

3 结语

多实例工作流模式是一类重要的工作流模式。本文利用 π 演算对多实例工作流模式进行了详细的描述，这种方法是完全形式化的，具有较强的语义表达能力，同时使过程模型的语义更加精确。运用 π 演算作为理论基础，形式化描述业务过程，能够方便地描述和分析工作流。

参考文献：

[1] MILNER R. The polyadic π - Calculus: A tutorial[A]. BAUER FL, BRAUER W, SCHWICHTENBERG H, eds. Logic and Algebra of Specification[C]. Berlin: Springer-Verlag, 1993. 203-246.

[2] MILNER R. Communicating and Mobile Systems: The π-calculus[M]. Cambridge: Cambridge University Press, 1999.

[3] SMITH H, FINGAR P. Business Process Management - The Third Wave[M]. Tampa: Meghan-Kiffer Press, 2002.

[4] VAN DER AALST WMP. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype"[EB/OL]. Http://is.tm.tue.nl/research/patterns/download/pi-hype.pdf, 2005-05-31.

[5] VAN DER ALAST WMP. Ter Hofstede AHM. Workflow Patterns[EB/OL]. Http://www.tm.tue.nl/it/research/patterns, 2003-03-10.

[6] LI CY, GOU J, WU HF, *et al.* A Process Meta-Model Supporting Domain Reuse[A]. 2005 International software process workshop[C]. 2005. 459-461.